metalogue

METALOGUE: Deliverable D2.2

Cognitive Model of the Game of Nines

The METALOGUE Consortium

November 2014

Version N°: [Version 2]

Main Authors: Christopher Stevens, Niels Taatgen, Fokie Cnossen

Project funded by the European Community under the Seventh Framework Programme for Research and Technological Development

Project ref. no.	ICT – 611073
Project title	METALOGUE – Multiperspective Multimodal Dialogue: dialogue system with metacognitive abilities

Document status	**Final version**
Contractual date of delivery	31 October 2014
Actual date of delivery	3 December 2014
Document number	D2.2
Deliverable title	Cognitive Model of the Game of Nines
Dissemination level	[Public]
Type	[Report]
Number of pages	[33]
WP contributing to the deliverable	WP 2
WP / Task responsible	University of Groningen
Contributing partners	University of Groningen
Reviewers	Internal Reviewer: Jan Alexandersson
Author(s)	Christopher Stevens, Niels Taatgen, Fokie Cnossen
EC Project Officer	Pierre-Paul Sondag
Keywords	Negotiation, Theory of Mind, Metacognition

Table of Contents

1. EXECUTIVE SUMMARY 5

2. INTRODUCTION 6

2.1 BACKGROUND AND TASK 7

2.2 EXPERIMENTAL PROCEDURES 7

2.3 AGENTS 7

4. THE COGNITIVE MODEL 10

4. SIMULATION RESULTS 10

4.1 COMPARISON OF STRATEGIES 13

4.2 COMPARISON WITH HUMAN DATA 13

4.3 DISCUSSION OF RESULTS 16

5. CONNECTION WITH THE METALOGUE SYSTEM 17

6 CONCLUSIONS 19

7. REFERENCES 20

8. ANNEXES 21

4.1 ANNEX 1. MODEL CODE 21

1. Executive Summary

In the present work, a cognitive model of the Game of Nines task is developed in the ACT-R cognitive architecture (Anderson, et al., 2004). This model is capable of negotiating with other agents in the Game of Nines task, and captures some general features of human data. This deliverable defines the Game of Nines task, presents a behavioral experiment using the task, describes the design philosophy and mechanisms of the Game of Nines model, and compares the output of the model to the behavioral data.

The results show that an agent with a metacognitive strategy that tries to predict the decisions of the opponent has a superior performance compared to an agent that uses either an aggressive or cooperative strategy. The performance of the model matches the performance of the best quartile of the human subjects, whereas most of the other subjects pursue a strategy that is too aggressive, and therefore miss the opportunity to maximise profit through cooperation.

We can therefore conclude that teaching a metacognitive strategy has a potential benefit for three quarters of our population.

2. Introduction

The aim of work package 2 is to create cognitive models capable of realistic, adaptive behaviour in negotiation scenarios. Here we present a cognitive model of a two-person bargaining game known as the Game of Nines (Kelley et al., 1967).

2.1 Scientific basis for strategy selection

A multitude of strategies and tactics are available to negotiators and vary widely depending on the context. However, two general classes of strategies often emerge in the literature: aggressive strategies and cooperative strategies. Aggressive strategies are characterized by high opening bids, resistance to making concessions, and a focus on maximizing one's own profit (Fisher, Ury, & Patton, 2001; Huffmeier, et al., 2014). Those using cooperative strategies, by contrast, tend to use mid-range opening bids, are more open to making concessions, and focus on agreement and consensus-building. Aggressive strategies tend to lead to better outcomes for the individual using them, but cooperative strategies lead to better socioemotional outcomes (e.g. the negotiator's partner will have a more favorable opinion of him/her when the negotiator uses a cooperative strategy) (Huffmeier et al., 2014). Skilled, professional negotiators should be able to identify and respond to both types of strategies. To this end, the model developed in the present work is designed to be capable of using and detecting both cooperative and aggressive strategies in negotiation.

2.2 The Game of Nines

The game of nines is a task first used by Kelley et al. (1967) to study negotiation behaviors in human participants. The game works as follows. Two players are given 9 points that must be split between them. In addition, each player has a minimum value that they must receive in order to not lose points (Minimum Necessary Share, or MNS). When a player receives points, their MNS is subtracted from the points to determine their profit. For instance, if a player's MNS is 2, and they receive 3 points, their profit for the round is 1. Each player knows only their own MNS value, not their opponent's. If the players cannot reach an agreement, neither player gets any points. The game is played over a series of rounds, each requiring the players to split up nine points. The structure of the negotiations is often open-ended, allowing the players to discuss potential deals as they wish.

Like many real-world negotiations, the game of nines is a mixed-motive situation. That is, players are motivated to work for the interests of the group but they also have interests that may conflict with those of the group. Both players have a clear incentive to reach an agreement that is acceptable to both. If the players are unable to reach an agreement, both receive nothing. However, players are also motivated to receive as many points as possible out of every deal. This can only be done at the expense of the other player. Many real-world negotiations are also mixed-motive. For example, suppose two parliament ministers are negotiating over how to appropriate funds. Both ministers likely have different ideas about

how they would like to spend the money, but if they fail to reach a consensus, a costly deadlock could occur. Similarly, a customer calling a call-center may desire a refund or replacement product, but the agent's job is to keep customers happy without costing the company too much money. Failure to reach an agreement here could be similarly costly for both parties.

2.3 Experimental procedures

Previous investigations using the Game of Nines have involved a two-person, open-ended setup. However, this by definition makes it impossible to control the strategies used by the players. Our goal here is to create a cognitive model that flexibly adapts to the strategies of multiple opponents in the same way that human players can. To this end, we designed an experiment in which one human player played against a computer agent with a fixed strategy. Two agents were developed. One agent attempted to be as fair as possible while the other simply tried to maximize its own gains. The behavior of both agents is described in greater detail below. We then had our cognitive agent play against the same computer opponents and compared the results.

2.3.1 Participants

Twenty-one people (8 male, age M = 21.8 SD = 2.1) participated in this experiment in exchange for 10 euros. This group consisted mainly of international students at The University of Groningen.

2.3.2 Procedure

All participants played against both agents in alternating order. Half of the participants played against the fair agent first and the other half played against the unfair agent first. Participants played for 6 blocks of 12 trials per block and the agent was switched every block. Each agent had a different name ("Tom" and "Ben") and a different line-drawing for their portrait. The names and portraits gave no indication about which agent was the "fair" agent. The names and portraits were selected so that each agent had the same "gender." This was intended to prevent the participants from behaving differently toward the agents on account of gender biases.

Participants interacted with the agent through a GUI run in Python 3.3 using the TkInter library (see Figure 1). At the beginning of every trial, both the player and the agent were shown their MNS values. The following pairs of MNS values were used: (2,2) (1,3) (3,1) (2,2) (3,3) (2,3) (3,2) (3,4) (4,3) (2,4) (4,2) (4,4). The order of these pairs was randomized for each participant and for each block. After the agent and player saw their MNS values, they were required to make a claim about their MNS values. These claims were visible to both and did not have to be true. Both the agent and the player could display dishonest MNS claims. But each player knew only their own MNS value and their opponent's claimed MNS

value. Then, the player was asked to make an offer to the agent. This offer indicated how many points the player wanted for him or her self. Any number from 1 to 9 could be selected. The player also had the option to indicate that this was his or her final offer. This indicated to the agent that the player would not make any lower offers. Finally, the player could quit the negotiation instead of making an offer. After the player made an offer, the agent would accept it, propose a counter offer, or quit. This process continued until an agreement was reached or one player quit. At the end of the trial, the player was notified of the outcome of the trial and the player's total number of points for the block was displayed.

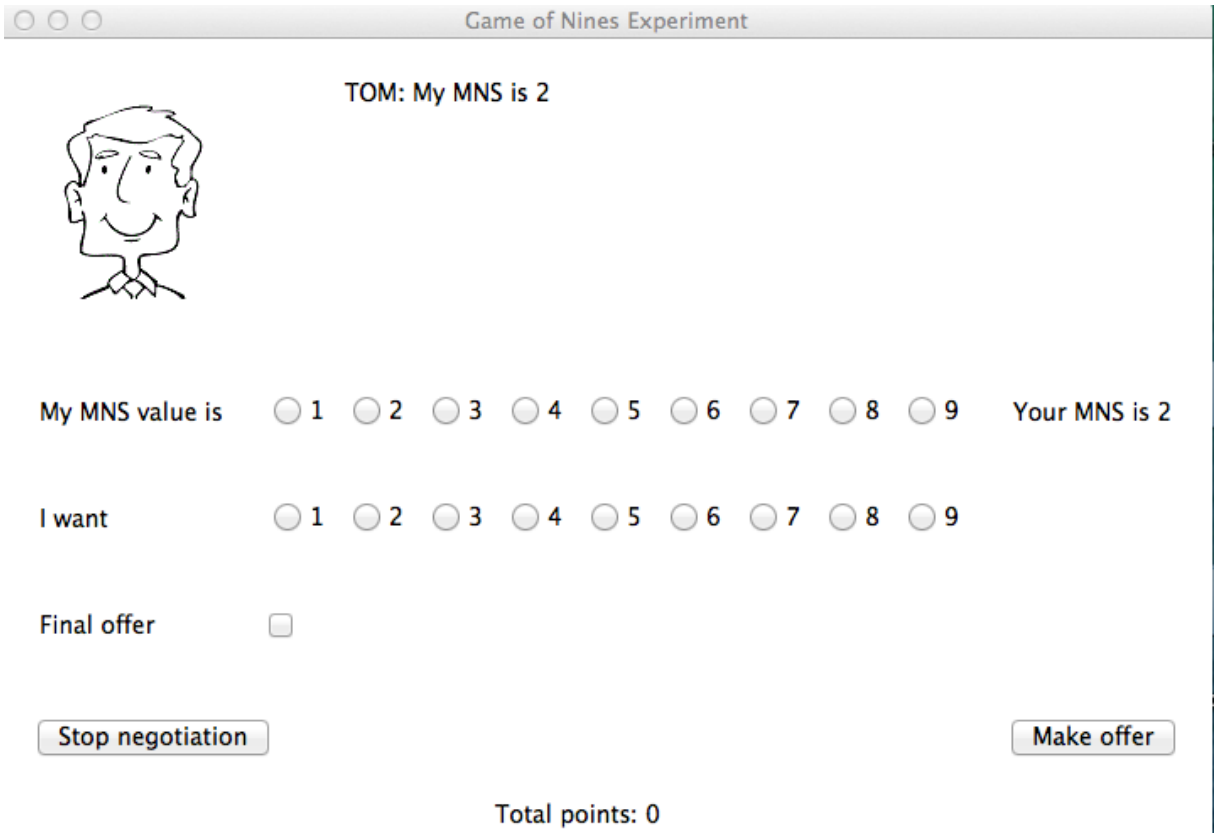

Figure 1. The user interface for the Game of Nines experiment

2.3.3 The Agents

The fair agent was designed to split the available points as evenly as possible between itself and the player. It computes the number of points leftover after subtracting its own MNS and the claimed MNS of its opponent. It then computes an offer that would result in both players profiting equally. However, it also monitors the player's MNS claims to prevent the player from taking advantage. To do this, it keeps a list in memory of all of its own past MNS values along with the player's past claims about their MNS values. The agent assumes that its MNS values on average are going to be the same as it's opponent's MNS values. If there is a discrepancy between the agent's MNS values, the agent will adjust the opponent's claimed MNS according to the following formula:

$$aMNS = round(cMNS - A * \frac{diffMNS}{se(MNSpa)})$$

Where:
aMNS: adjusted MNS
A: a constant representing the weighting of the trust value. For the fair agent, this is set to 1.
cMNSp: The player's claimed MNS value
diffMNS: The difference between the means of the agent's MNS values and the claimed values of the player
MNSpa: A combined list of the MNS values of the agent and the player's claimed values
se(MNSpa): The standard error in the MNS values, which is an indication of uncertainty that decreases with more experience

In other words, the agent generates an estimate of whether and how much its opponent is lying, then it adjusts the opponent's claim by that amount. As a result, the agent will play more aggressively when the player lies about his or her MNS.

The unfair agent, by contrast, was designed to maximize its own profit. The agent routinely lies about its own MNS value based on the following formula:

$$cMNS = MNS + round(\frac{4}{MNS})$$

The unfair agent uses the same formula as the fair agent to calculate a fair offer with two exceptions. First, it will use its claimed MNS value rather than its actual value. This results in the agent making more aggressive offers on average than the fair agent. Second, it does not weigh its opponent's lies as heavily (the constant, A, is set to .5 rather than 1).

Both agents will accept an offer only if it is at least as good as their computed "fair offer." If the player asks for more than this, the agent will propose a counter offer using the following formula:

$$\text{offer}_{agent} = \text{fair}_{agent} + (\text{offer}_{player} + \text{fair}_{player})$$

Where *fair* denotes the agent's calculation of how many points the agent and the player should receive respectively. In other words, the agent will make an offer that is just as high above its own fair value as the opponent's offer is above their fair value.

One final difference between the two agents is how they handle a player's final offer. The fair agent has a small chance of accepting an offer higher than its fair offer. But it will not accept anything 3 or more points above the fair offer. When the unfair agent receives a final offer, it considers how many points it will gain. This agent has a high probably of accepting a final offer as long as it will make a profit of at least one point. Thus, the use of the final offer option is much more effective against this agent.

3. The Cognitive Model

We developed a model in the ACT-R cognitive architecture capable of playing the game of nines. This model uses theory-of-mind (see Premack & Woodruff, 1978; Gallese & Goldman, 1998) to reason about its opponent's strategy and deploys a strategy of its own based on this reasoning. The majority of the functionality of the model comes from activation of declarative knowledge within ACT-R's declarative memory system. In the following section, the principles that govern declarative memory in ACT-R will be described. Then, the strategies employed by the model (aggressive and cooperative) will be defined. Finally, the model's theory-of-mind reasoning process will be explained.

3.1 Declarative Memory in ACT-R

ACT-R's declarative memory consists of a network of schematic units known as chunks. Each chunk has one or more slots that contain values or links to other chunks in declarative memory. Each chunk also possesses an activation value. More active chunks are more likely to be retrieved in a search of declarative memory. The activation level of a chunk (i) is derived from the following equation.

$$A_i = \left(\ln\left(\frac{n}{1-d}\right) - d * \ln(L)\right) + N\left(0, \frac{\pi * s}{\sqrt{3}}\right)$$

In this equation, n is the number of presentations of chunk i. L represents the amount of time that has passed since the creation of the chunk and d is the rate of activation decay. The rightmost term of the equation represents noise added to the activation level. This equation dictates that the activation level of a chunk is determined by how frequently it is retrieved from memory. In the standard ACT-R activation equation, activation also depends on how recently the chunk has been retrieved. However, here we use a simplified equation that assumes all references to the chunk are uniformly distributed throughout the life of the chunk.

A chunk does not have to be a perfect match to a retrieval request in order to be retrieved. When a chunk in declarative memory is not a perfect match to a retrieval request, then ACT-R will reduce its activation according to the following formula:

$$Similarity = \sum_{l} PM_{li}$$

P denotes the mismatch penalty. When P is higher, activation is more strongly affected similarity. In this model it is set to 5. M indicates the similarity value between the relevant slot value in the retrieval request (*l*) and the corresponding slot chunk *i* (the chunk in declarative memory) summed over all slot values in the retrieval request.

Research on human number representation has shown that discrimination between numerical quantities obeys Weber's Law (Whalen, Gallistel, & Gelman, 1999). Therefore, the similarity between numerical slot values can be given by the following equation:

$$M_{li} = -\frac{abs(l-i)}{\max(l,i)}$$

According to this equation, numbers that are closer together will be considered more similar. In addition, larger pairs of numbers will be more similar than smaller pairs.

Thus the model can retrieve chunks that are not exact matches, but the mismatch penalty makes this less likely. This has two important implications for the current model. First, it means that the model will be able to retrieve past instances for reasoning even when the model has not encountered a particular situation before. Partial matching, combined with activation noise, also allows for flexibility in the model's behaviour. It will not rigidly make the exact same moves every time.

3.2 Strategies

There are two major decisions a player needs to make when playing a round of the game of nines. First, what should the opening offer be? Aggressive negotiators will give high opening offers while cooperative negotiators will give more moderate offers (Huffmeier et al., 2014). The opening offer is important because it sends the partner a message about the player's goal for the round. A higher opening offer suggests that the player may have a higher MNS. Alternatively, a player could "bluff," giving a high opening offer regardless of MNS. People often anchor their goals for negotiation on their partner's opening offer, and will ask for less when the opening offer is high. However, this will backfire if they realize their partner is bluffing. In this case, people become more demanding in their counteroffers (Liebert, et al., 1968).

The second major decision a player must make is the range of offers he or she will accept. What is the minimum acceptable gain from the deal? When will the player quit or threaten to quit? This may be different than the player's MNS value. For instance, a player may be unwilling to consider any deal that does not earn him or her at least one point. So if this player's MNS value is 3, they will only accept an offer of at least 4. In general, aggressive players are more demanding and will have higher minimum gains than cooperative players.

The cognitive model described here can use both cooperative and aggressive strategies flexibly. Both strategies are coded into the model's declarative memory. There are two types of chunks: chunks that specify opening offers and chunks that specify minimum goals. These chunks have slots that code the type of strategy they correspond to (cooperative or aggressive) and the model's MNS value. At the beginning of a round, the model searches declarative memory based on which strategy is currently active and the current MNS value to

select an opening offer and a minimum goal. A list of the values used in these strategies can be found in Table 1.

Table 1. Comparison of Cooperative and Aggressive Strategies

Strategy	MNS	First-offer	Minimum-offer
Cooperative	1	4	1
	2	4	2
	3	5	3
	4	5	4
Aggressive	1	8	5
	2	8	5
	3	9	6
	4	9	6

3.3 Theory-of-mind

The model attempts to infer the strategy its opponent is using and selects its own strategy accordingly. If it detects a cooperative strategy, the model will play cooperatively. However, the cooperative strategy can be easily exploited by an aggressive opponent. Therefore, if the model detects that its opponent is playing aggressively, then it will choose the aggressive strategy as well.

To reason about its partner's strategy, the model employs a style of theory-of-mind similar to the "simulation" style of theory of mind (Gallese & Goldman, 1998). That is, the model assumes that the opponent would make decisions similar to itself. The model monitors both the opening offers and last offers made by the opponent. Based on these values, the model retrieves its own strategy chunks from memory. If these strategy chunks are both "aggressive" style chunks, the model will conclude that the opponent is playing aggressively and choose an aggressive strategy. Otherwise, the model will choose a cooperative strategy.

Consider the following illustrative example. On a given round, the unfair agent plays 8 as its first offer. At the end of negotiation, an agreement is finally reached in which the unfair agent receives 5 points. Thus the model retrieves the following chunks from memory:

(aggrfirst2 isa first-offer player me strategy aggressive mns 2 offer 8)

(aggrlast2 isa last-offer player me strategy aggressive mns 2 offer 5)

From this the model concludes that its partner is playing the same way it would if it was using

an aggressive strategy and its MNS was 2. It responds by selecting the aggressive strategy for the following round.

The reasoning approach described above does not account for the model's behavior during a given round. If the model chooses an aggressive strategy, even a generally cooperative opponent may play defensively by making aggressive offers. This may lead the model to incorrectly conclude that the opponent is generally aggressive. Indeed, early pilot runs indicated that the model would gradually shift toward an aggressive strategy, even with the fair agent. To prevent this, the current model gives extra weight to chunks in which it played a cooperative strategy. Chunks with the value "aggressive" in the "strategy" slot receive a mismatch penalty according to the similarity equation described above.

In the Game of Nines task, people rarely make explicit claims about their MNS values unless prompted by the experimenter. Thus this model does not include any explicit strategies regarding MNS claims. By default, it will make an honest claim about its MNS value if prompted. However, in the human data, we observed that people often overstate their MNS values by at least one. Therefore, for the purposes of matching participant performance in this dataset, the model will randomly select a number between 0 and 2 to add to its MNS claim when prompted. Lying about MNS values may be useful to implement as a strategy choice in later versions of the model, but it is beyond the scope of the current model.

4. Simulation Results

4.1 Strategy Comparison

Our first aim was to validate that the cooperative and aggressive strategies really do perform better against the fair and unfair agents respectively. In addition, we aimed to verify that flexibly using both strategies does in fact provide an advantage. To test this, we compared three different versions of the model: one that used only the cooperative strategy, one that used only the aggressive strategy, and one that used both (metacognitive). We played each model against the fair and unfair agents 100 times. The results can be found in Table 2.

We used two different performance measures to compare the strategies: model score and disparity. Model score is simply the average score of the model. And disparity is the distance between the model's score and the agent's score. A negative disparity means that the agents scored higher on average. As expected, the cooperative strategy performed better against the fair agent and the aggressive strategy performed better against the unfair agent. However, the performance of both strategies dropped sharply when paired with the other agent. This is also evident in the disparity scores. Also, the cooperative model managed to reach more agreements than the other models, but these agreements were very uneven. The opponents of the cooperative model scored much more highly than the cooperative model did, suggesting the cooperative model yielded too much in negotiations. This means that on average, the metacognitive model was able to achieve better outcomes for itself and more equitable outcomes overall than the single-strategy models.

Table 2. Individual model scores and disparity scores for each individual strategy and the combined metacognitive strategy

Model Score		
	Opponent	
Strategy	Fair	Unfair
Aggressive	9	16
Cooperative	15	9
Metacognitive	15	14
Disparity		
	Opponent	
Strategy	Fair	Unfair
Aggressive	4.2	8.2
Cooperative	-12	-24
Metacognitive	-3	1

4.2 Comparison with Human Data

In order to identify differences in strategy between lower and higher performing subjects, we

split the performance data into 4 quartiles based on overall score. These data can be seen in Figure 2. Of particular interest here is the comparison between the middle 50 percent of subjects and the top 25 percent. The middle subjects performed better against the unfair agent than the fair agent. However, the top subjects performed well against both agents. We propose that the reason for the difference in performance is that the middle subjects apply aggressive strategies more quickly than the top subjects. To capture this in the model, we made two changes. First, when searching declarative memory, the middle subject model does not have a bias for cooperative trials. The middle subjects may lack the metacognitive knowledge that otherwise cooperative opponents will probably respond aggressively to aggressive tactics. Second, the middle-subject model uses a modified decision rule. The top-subject model will always choose to cooperate if the opponent's strategy is ambiguous, but the middle-subject model will randomly choose between the cooperative and aggressive strategy in this case.

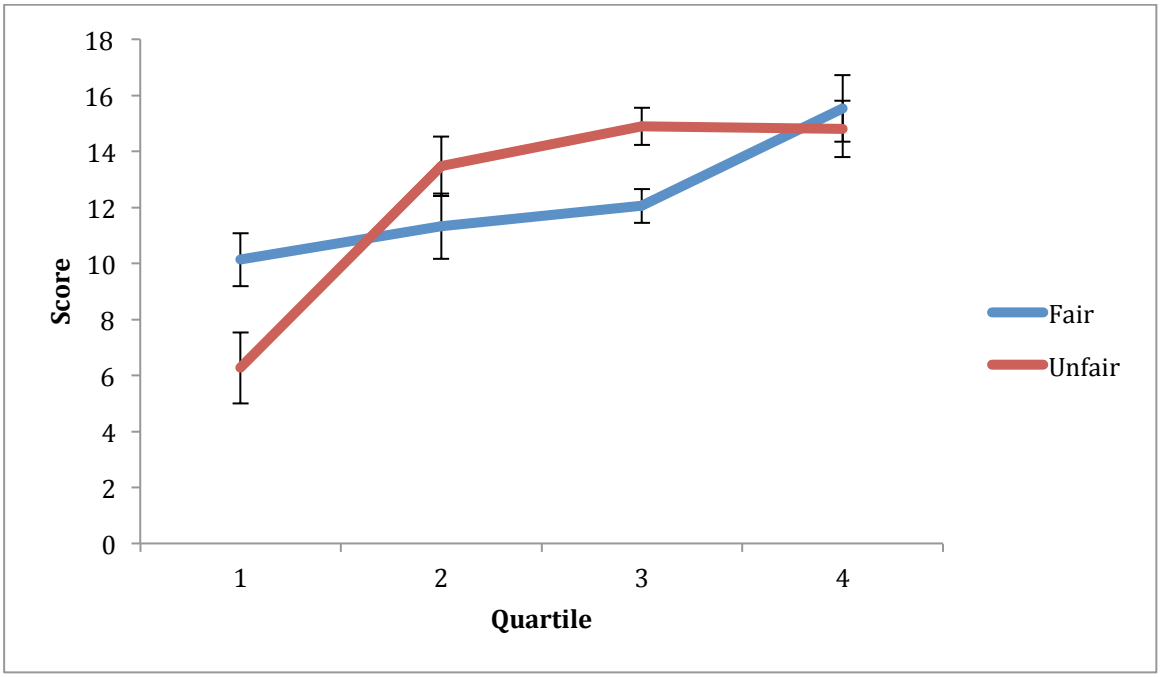

Figure 2. Scores of human subjects by agent and quartile.

The fair and unfair agents were ported into a lisp environment to allow interaction with the model. The experimental procedure described above was replicated in the environment. During each run, the model played against the agents for a total of six blocks, alternating between the agents after each block. The model was reset between runs. The model was run 200 times. In half the runs, the model played against the unfair agent first and in the other half it played against the fair agent first. The model does not yet simulate strategy learning. So we fit the model to data from subjects who already demonstrated skill in the task. We used the top 50% of subjects in terms of overall score. The fit of the model to the data can be seen in Figure 3. As shown in Figure 3, the model provides a good fit for both the third and fourth quartile subjects across both agents (RMSE = .47, R = .96).

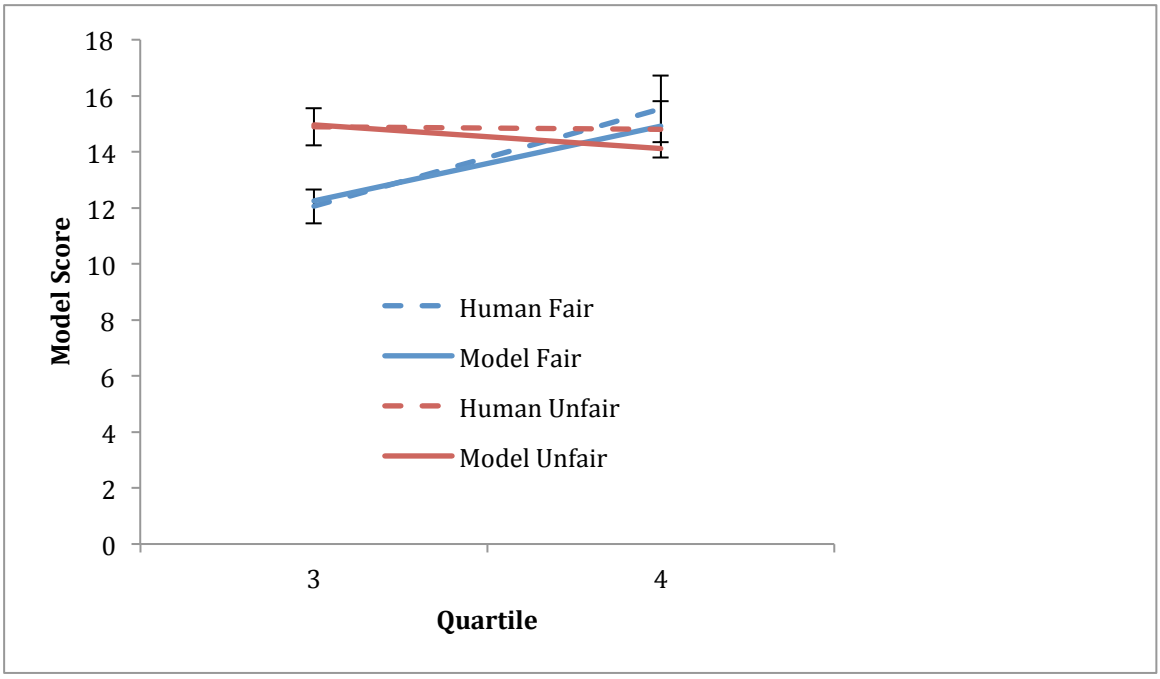

Figure 3. Scores of human subjects by agent and quartile.

The model also replicates another important feature of the behavioral data: the use of final-offers. Both the third and fourth quartile subjects show a strong tendency to use the final-offer move very frequently against the unfair agent (Third = 74.5%, Fourth = 73.9%). However, the third quartile subjects used final offers against the fair agent much more frequently than the fourth quartile subjects did (Third = 63%, Fourth = 53%). The model provides a good qualitative fit to this trend, as can be seen in Figure 4.

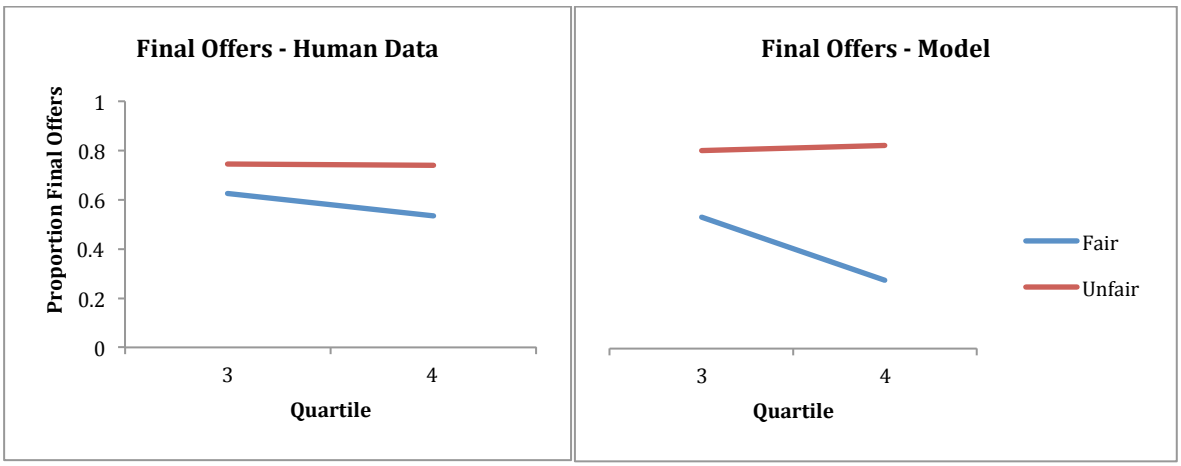

Figure 4. Proportion of final offers in the data and model.

It is also apparent from Figure 4 that the current model overpredicts the magnitude of the difference in final offers between the fair and unfair agents. This could be because the current model always uses the final offer move when it reaches its minimum goal. This suggests that people use the final offer move more probabilistically than the model. Perhaps

a better fit could be obtained by adding chunks to the model that specify when to use the final offer move. These chunks could be setup so that they are more likely to be selected when the model approaches its minimum, but are not guaranteed to be selected, even if the minimum is reached.

4.3 Discussion of results

The results reported here suggest that metacognition does improve performance in a negotiation task, but many human subjects do not use it effectively. Specifically, the best subjects in this task were the ones that could identify the best strategy to use against each opponent. As a result, these subjects were able to perform well against cooperative and aggressive opponents. By contrast, lower-scoring subjects tended to use an aggressive strategy when a cooperative strategy would have been more appropriate. This resulted in lower overall performance. The bottom quartile of subjects proved most difficult to capture with the model. The participants in this quartile likely applied a variety of ineffective strategies. Most of the sample showed suboptimal metacognition, so there is much room for improvement through training.

These results reveal an area in which training with the METALOGUE system could be very useful. To become better negotiators, students should learn to identify different types of opponents and the strategies that they use. They must also learn which strategy to apply against each type of opponent. We believe that a very efficient way to accomplish this learning is by having students practice against agents designed to mimic different negotiation strategies, and provide feedback if they do not use the appropriate strategy for that agent.

Although we discussed only cooperative and aggressive strategies, we do not claim that these are the only two strategies people might use in this task. For instance, one might use a neutral strategy and try not to be overly cooperative or aggressive. Alternatively, one could also apply a utilitarian strategy and simply try to make the moves that result in the highest payoffs.

5. Further development and connection with the METALOGUE system

The model presented here is the first step toward an agent capable of realistic, adaptive negotiation. The model is capable of performing the game of nines task at a level similar to skilled human players, and demonstrates how cooperative and aggressive strategies can be selected dynamically to effectively manage different types of opponents. Furthermore, the model successfully replicates the performance of human participants at different levels of skill.

It is important to note that the current negotiation context was very constrained. As a result, the model is not yet capable of simulating human verbal tactics. There are several types of tactics commonly employed by people in negotiation tasks like the game of nines. We have previously collected a set of human data in which subjects were permitted to discuss their offers however they wished. We documented several common verbal strategies, including promises of more generous offers on future rounds, (unprompted) lies about MNS values, accusations of cheating, and threats to quit. Future versions of the model could include these tactics. In principle, the tactics could be implemented as declarative memory chunks just like opening offers and minimum goals. When interacting with humans, it is also possible to make the model produce simple, fixed verbal responses (e.g. "You have been getting a lot of points. I don't think you are playing fair"). However, it is beyond the scope of the model to interpret or produce human dialogue.

In the present experiment, both players had to make explicit claims about their MNS values. As previously noted, this is not a common feature of many negotiation tasks. However, the functioning of the current model is not dependent on this feature. In fact, the model's metacognitive reasoning ignores opponents' MNS claims altogether. Therefore, the model could readily be applied to a variety of negotiation contexts.

These cognitive models of the game of nines make two important contributions to the METALOGUE system. First, we believe the models produced as a part of this work package will be very useful within the METALOGUE system as virtual opponents against which players can practice negotiation. Although players cannot yet interact with the model, deliverable 2.4 will introduce a user interface that students may use to play the game of nines against the model. Further development is necessary before the model can simulate virtual opponents in the pilot scenarios. Of utmost importance is the creation of a system by which the utility of prospective agreements to both parties can be measured. The current model works with points. It will be necessary to find a way to map those points onto positions in the context of a parliamentary debate or call center interaction.

The second contribution of this work is the insights revealed by fitting the model to human data. By doing so, we have provided evidence that metacognitive reasoning can indeed improve performance in negotiation, but that only a fraction of people use it successfully.

This supports the assumption of the METALOGUE system that metacognitive skills are valuable to negotiators and should be trained. Additionally, the model does not only monitor the opponent's behavior, but also accounts for the fact that its own behaviors influence its opponent. This suggests that an important skill for negotiators to learn is how their own behaviors influence the actions of their opponent. For example, if a parliament member refuses to compromise and makes very high demands, other members may be unwilling make concessions as well. But it would be a mistake for the first member to conclude that the other members are simply being stubborn and unreasonable.

6. Conclusions

The cognitive model presented in this deliverable provides a good account of human behavior in the game of nines task as well as a foundation for developing artificial opponents for people to train against. Fitting the human data suggests that only a few subjects use metacognition effectively in this task, so many could benefit from training in metacognitive skills. Specifically, students would benefit from training in how to identify and apply appropriate strategies for a given opponent.

7. References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036.

Gallese, V., & Goldman, A. (1998). Mirror neurons and the simulation theory of mind-reading. *Trends in Cognitive Sciences*, *2*(12), 493–501.

Fisher, R., Ury, W. L., & Patton, B. (2011). *Getting to yes: Negotiating agreement without giving in*. Penguin.

Huffmeier, J., Freund, P. A., Zerres, A., Backhaus, K., Hertel, G., (2014). Being tough or being nice? A meta-analysis on the impact of hard- and softline strategies in distributive negotiations. *Journal of Management, 40,* 866-92.

Kelley, H., Beckman, L., & Fischer, C. (1967). Negotiating the division of a reward under incomplete information. *Journal of Experimental Social Psychology*, *361-98*.

Liebert, R., Smith, W., Hill, J., & Keiffer, M. (1968). The effects of information and magnitude of initial offer on interpersonal negotiation. *Journal of Experimental Social Psychology*, *4*, 431–41.

Premack, D., & Woodruff, G. (1978). Does the chimpanzee have a theory of mind? *Behavioral and Brain Sciences*, 515–526.

Whalen, J., Gallistel, C. R., & Gelman. (1999). Nonverbal counting in humans: The psychophysics of number representation. *Psychological Science, 10,* 130-7.

8. Annex 1: Model Code

```
(define-model nines1

(sgp :esc t :ol t :er t :bll .5  :v nil :rt -60 :sim-hook number-sims :mp 5 :trace-detail low :lf 0 )

(chunk-type state opponent my-mns my-claim opp-mns my-final opp-final my-offer opp-offer min outcome points status strategy)
(chunk-type mns opponent mine opp diff)
(chunk-type opp-first-offer opponent value)
(chunk-type opp-last-offer opponent value)
(chunk-type opponent name first-offer final-offer)
(chunk-type trial opp-first-offer opp-last-offer outcome points)
(chunk-type outcome action points)
(chunk-type first-offer player strategy mns offer)
(chunk-type last-offer player strategy mns offer)
(chunk-type strategy)
(chunk-type game my-strategy opp-strategy opponent my-first-offer opp-first-offer my-last-offer opp-last-offer)
(chunk-type round player strategy mns first-offer last-offer)
(chunk-type claim player strategy mns claim)

(add-dm
 (coop isa strategy)
 (aggr isa strategy)
 (neutral isa strategy)
 (coopfirst1 isa first-offer player me strategy coop mns 1 offer 4)
 (coopfirst2 isa first-offer player me strategy coop mns 2 offer4)
 (coopfirst3 isa first-offer player me strategy coop mns 3 offer 5)
 (coopfirst4 isa first-offer player me strategy coop mns 4 offer 5)
 (aggrfirst1 isa first-offer player me strategy aggr mns 1 offer 8)
 (aggrfirst2 isa first-offer player me strategy aggr mns 2 offer 8)
 (aggrfirst3 isa first-offer player me strategy aggr mns 3 offer 9)
 (aggrfirst4 isa first-offer player me strategy aggr mns 4 offer 9)
 (cooplast1 isa last-offer player me strategy coop mns 1 offer 1)
 (cooplast2 isa last-offer player me strategy coop mns 2 offer 2)
 (cooplast3 isa last-offer player me strategy coop mns 3 offer3)
 (cooplast4 isa last-offer player me strategy coop mns 4 offer 4)
 (aggrlast1 isa last-offer player me strategy aggr mns 1 offer 5)
 (aggrlast2 isa last-offer player me strategy aggr mns 2 offer 5)
 (aggrlast3 isa last-offer player me strategy aggr mns 3 offer 6)
 (aggrlast4 isa last-offer player me strategy aggr mns 4 offer 6)
 ;(coop1 isa round player me strategy coop mns 1 first-offer 4 last-offer 2)
 ;(coop2 isa round player me strategy coop mns 2 first-offer 5 last-offer 3)
 ;(coop3 isa round player me strategy coop mns 3 first-offer 6 last-offer 4)
 ;(coop4 isa round player me strategy coop mns 4 first-offer 6 last-offer 5)
 ;(aggr1 isa round player me strategy aggr mns 1 first-offer 7 last-offer 4)
 ;(aggr2 isa round player me strategy aggr mns 2 first-offer 7 last-offer 5)
 ;(aggr3 isa round player me strategy aggr mns 3 first-offer 8 last-offer 6)
 ;(aggr4 isa round player me strategy aggr mns 4 first-offer 9 last-offer 7)
)

(sdp (coopfirst1 :reference-count 10))
(sdp (coopfirst2 :reference-count 10))
(sdp (coopfirst3 :reference-count 10))
(sdp (coopfirst4 :reference-count 10))
(sdp (aggrfirst1 :reference-count 10))
(sdp (aggrfirst2 :reference-count 10))
(sdp (aggrfirst3 :reference-count 10))
(sdp (aggrfirst4 :reference-count 10))
(sdp (cooplast1 :reference-count 10))
(sdp (cooplast2 :reference-count 10))
(sdp (cooplast3 :reference-count 10))
(sdp (cooplast4 :reference-count 10))
(sdp (aggrlast1 :reference-count 10))
(sdp (aggrlast2 :reference-count 10))
(sdp (aggrlast3 :reference-count 10))
(sdp (aggrlast4 :reference-count 10))


(p new-game
   =goal>
      isa         state
      opponent    =opponent
      my-claim    0
```

```
      my-mns      =mymns
      my-offer    0
      status      active
      outcome     nil
   ?imaginal>
      state       free
      buffer      empty
==>
  !bind! =claim (+ =mymns 0)
  ;!bind! =strat (if (= (act-r-random 2) 1) 'aggr 'coop) ;use this to make the model randomly select its strategy
  !bind! =strat 'coop
  +imaginal>
      isa             game
      opponent        =opponent
      my-strategy     =strat
  =goal>
      my-claim    =mymns
  !eval! (if (eql =strat 'coop) (setf *coop* (+ 1 *coop*)))
  !eval! (if (eql =strat 'aggr) (setf *aggr* (+ 1 *aggr*)))
)

(p switch-opponent
   =goal>
      isa         state
      opponent    =opponent
      my-claim    0
      my-mns      =mymns
      my-offer    0
      status      active
      outcome     nil
    =imaginal>
      isa             game
      - opponent        =opponent
    ?retrieval>
      state           free
      buffer          empty
==>
    +retrieval>
        isa           game
        opponent      =opponent
    =imaginal>
)

(p remember-opponent
  =goal>
      isa         state
      opponent    =opponent
      my-claim    0
      my-mns      =mymns
      my-offer    0
      status      active
      outcome     nil
    =imaginal>
      isa             game
      - opponent        =opponent
    =retrieval>
      isa             game
      opponent        =opponent
      my-strategy     =strat
==>
   ;!bind! =strat (if (= (act-r-random 2) 1) 'aggr 'coop) ;use this to make the model randomly select its strategy
   ;=imaginal>
   ;    opponent        nil ;added to correct hanging Nov10
   ;-imaginal>
   =imaginal>
      ;isa             game
      opponent        =opponent
      my-strategy     =strat
  =goal>
      my-claim    =mymns
  !eval! (if (eql =strat 'coop) (setf *coop* (+ 1 *coop*)))
  !eval! (if (eql =strat 'aggr) (setf *aggr* (+ 1 *aggr*)))
)

(p new-opponent
  =goal>
```

```
      isa         state
      opponent    =opponent
      my-claim    0
      my-mns      =mymns
      my-offer    0
      status      active
      outcome     nil
    =imaginal>
       isa             game
       - opponent        =opponent
    ?retrieval>
       state           error
==>
   ;!bind! =strat (if (= (act-r-random 2) 1) 'aggr 'coop) ;use this to make the model randomly select its strategy
   !bind! =strat 'coop
   ;=imaginal>
   ;    opponent        nil ;added to correct hanging Nov10
   ;-imaginal>
   =imaginal>
      ; isa             game
       opponent        =opponent
       my-strategy     =strat
  =goal>
       my-claim   =mymns
  !eval! (if (eql =strat 'coop) (setf *coop* (+ 1 *coop*)))
  !eval! (if (eql =strat 'aggr) (setf *aggr* (+ 1 *aggr*)))
)

(p new-opponent-2
  =goal>
      isa         state
      opponent    =opponent
      my-claim    0
      my-mns      =mymns
      my-offer    0
      status      active
      outcome     nil
    =imaginal>
       isa             game
       - opponent        =opponent
    =retrieval>
       isa             game
       - opponent      =opponent
==>
   ;!bind! =strat (if (= (act-r-random 2) 1) 'aggr 'coop) ;use this to make the model randomly select its strategy
   !bind! =strat 'coop
   ;=imaginal>
   ;    opponent        nil ;added to correct hanging Nov10
   ;-imaginal>
   =imaginal>
      ; isa             game
       opponent        =opponent
       my-strategy     =strat
  =goal>
       my-claim   =mymns
  !eval! (if (eql =strat 'coop) (setf *coop* (+ 1 *coop*)))
  !eval! (if (eql =strat 'aggr) (setf *aggr* (+ 1 *aggr*)))
)



(p new-trial
   =goal>
      isa         state
      opponent    =opponent
      my-claim    0
      my-mns      =mymns
      my-offer    0
      status      active
      outcome     nil
  =imaginal>
       isa             game
       opponent        =opponent
       my-strategy     =strat
==>
  =goal>
```

```
      my-claim          =mymns
  -retrieval>
  =imaginal>
   ;    my-first-offer    nil
   ;    opp-first-offer   nil
   ;    my-last-offer     nil
   ;    opp-last-offer    nil

   !eval! (if (eql =strat 'coop) (setf *coop* (+ 1 *coop*)))
   !eval! (if (eql =strat 'aggr) (setf *aggr* (+ 1 *aggr*)))

)



(p read-first-offer
  =goal>
      isa          state
      status       active
      > opp-offer  0
      opp-offer    =offer
  =imaginal>
      isa                  game
      opp-first-offer      nil
      opp-last-offer       nil
==>
  =imaginal>
      opp-first-offer   =offer
      opp-last-offer    =offer
)

(p read-new-offer
  =goal>
      isa          state
      status       active
      > opp-offer  0
      opp-offer    =offer
  =imaginal>
       isa                  game
       > opp-first-offer    0
      - opp-last-offer      =offer
==>
  =imaginal>
      opp-last-offer   =offer
)

(p select-first-offer-1
  =goal>
      isa          state
      status       active
      outcome      nil
      my-mns       =mymns
      > my-claim     0
      my-offer     0
  =imaginal>
      isa                  game
      my-strategy          =strat
      my-first-offer       nil
      my-last-offer        nil
  ?retrieval>
      state      free
      buffer     empty
==>
  +retrieval>
      isa       first-offer
      player    me
      strategy  =strat
      mns       =mymns
  =imaginal>
   ;    my-first-offer      pending
)

(p select-first-offer-2
  =goal>
      isa          state
      status       active
```

```
      outcome      nil
      my-mns       =mymns
      > my-claim     0
      my-offer      0
  =imaginal>
      isa                 game
      my-strategy          =strat
      my-first-offer       nil
      my-last-offer        nil
  =retrieval>
      isa           first-offer
      ;mns           =mymns
      offer          =offer
==>
  =imaginal>
      my-first-offer       =offer
  ;=retrieval>
  ;     mns      nil
  ;     offer    nil
  ;-retrieval>
)

(p select-last-offer-1
   =goal>
      isa          state
      status       active
      outcome      nil
      my-mns       =mymns
      > my-claim     0
      my-offer      0
  =imaginal>
      isa                 game
      my-strategy          =strat
      > my-first-offer      0
      my-last-offer        nil
  ?retrieval>
      state     free
      buffer    empty
==>
   +retrieval>
      isa       last-offer
      player    me
      strategy  =strat
      mns       =mymns
   =imaginal>
    ;   my-last-offer     pending
)

(p select-last-offer-2
   =goal>
      isa          state
      status       active
      outcome      nil
      my-mns       =mymns
      > my-claim     0
      my-offer      0
  =imaginal>
      isa                 game
      my-strategy          =strat
      > my-first-offer      0
      my-last-offer        nil
  =retrieval>
      isa           last-offer
      ;mns           =mymns
      offer          =offer
==>
  =imaginal>
      my-last-offer       =offer
  ;=retrieval>
  ;     mns      nil
  ;     offer    nil
  ;-retrieval>

)

(p make-first-offer
```

```
   =goal>
      isa         state
      status      active
      outcome     nil
      > my-claim   0
      my-offer    0
  =imaginal>
      isa                 game
      my-strategy         =strat
      > my-first-offer    0
      my-first-offer      =first
      > my-last-offer     0
==>
  =goal>
     my-offer       =first
     status         wait
  =imaginal>
)


(p accept-offer
   =goal>
      isa         state
      status      active
      outcome     nil
      > my-claim   0
      > opp-offer  0
      opp-offer   =oppoffer
  =imaginal>
      isa                 game
      my-strategy         =strat
      my-first-offer      =first
      my-last-offer       =last
      opp-last-offer      =oppoffer


!bind! =mypoints (- 9 =oppoffer)
!eval! (> =mypoints =last)

==>
   =goal>
      my-offer        =mypoints
      status          wait
   =imaginal>

)



(p make-counter-offer
   =goal>
      isa         state
      status      active
      outcome     nil
      > my-claim   0
      > opp-offer  0
      opp-offer   =oppoffer
      > my-offer   0
      my-offer    =myoffer
  =imaginal>
      isa                 game
      my-strategy         =strat
      my-first-offer      =first
      my-last-offer       =last
      opp-first-offer     =oppfirst
      opp-last-offer      =oppoffer

!bind! =mypoints (- 9 =oppoffer)
!eval! (not (> =mypoints =last))
!eval! (> =myoffer =last)
==>
!bind! =newoffer (- =myoffer 1)

   =goal>
       my-offer     =newoffer
       status       wait
```

```
  =imaginal>
)



(p final-offer
  =goal>
      isa          state
      status       active
      outcome      nil
      > my-claim   0
      > opp-offer  0
      opp-offer    =oppoffer
      > my-offer   0
      my-offer     =myoffer
  =imaginal>
      isa                  game
      my-strategy          =strat
      my-first-offer       =first
      my-last-offer        =last
      opp-last-offer       =oppoffer

!bind! =mypoints (- 9 =oppoffer)
!eval! (not (> =mypoints =last))
!eval! (not (> =myoffer  =last))
==>
  =goal>
       my-final      1
       status        wait
  =imaginal>

)

(p quit
  =goal>
      isa          state
      status       active
      outcome      nil
      > my-claim   0
      > opp-offer  0
      opp-offer    =oppoffer
      > my-offer   0
      my-offer     =myoffer
      my-final     1
  =imaginal>
      isa                  game
      my-strategy          =strat
      my-first-offer       =first
      my-last-offer        =last
      ;opp-last-offer       nil

==>
  =goal>
       my-offer      -1
       status        wait
  =imaginal>

)

(p store-last-offer
   =goal>
      isa          state
      status       active
      outcome      =outcome
      points       =points
      opp-offer    =oppoffer
  =imaginal>
      isa              game
      opp-first-offer    =first
      opp-last-offer     nil
==>
  =imaginal>
      opp-last-offer     =oppoffer

)
```

```
;(p first-round-quit
;   =goal>
;       isa       state
;       status    active
;       outcome   nil

;)

;;Learning productions

(p eval-opp-first-offer-1
   =goal>
      isa         state
      status      active
      outcome     =outcome
      points      =points
   =imaginal>
      isa              game
      opp-first-offer    =first
      - opp-first-offer  coop
      - opp-first-offer  aggr
      opp-last-offer     =last
      - opp-last-offer   coop
      - opp-last-offer   aggr
   ?retrieval>
      state      free
      buffer     empty
==>
   +retrieval>
      isa        first-offer
      player     me
      offer      =first
   =imaginal>
)

(p eval-opp-first-offer-2
    =goal>
      isa         state
      status      active
      opponent    =opponent
      outcome     =outcome
      points      =points
   =imaginal>
      isa              game
      opp-first-offer     =first
      - opp-first-offer   aggr
      - opp-first-offer   coop
      opp-last-offer      =last
      - opp-last-offer    coop
      - opp-last-offer    aggr
   =retrieval>
      isa         first-offer
     ; player      me
       strategy    =strat
==>
   =retrieval>
      player      =opponent
   -retrieval>
   =imaginal>
      opp-first-offer    =strat
   +retrieval>
      isa        last-offer
      player     me
      offer      =last
)
#|
(p eval-opp-last-offer-1
   =goal>
      isa         state
      status      active
      outcome     =outcome
      points      =points
   =imaginal>
      isa              game
      opp-first-offer    done
```

```
      - opp-last-offer   done
      opp-last-offer     =last
  ?retrieval>
      state       free
      buffer      empty
==>
  +retrieval>
      isa         last-offer
      player      me
      offer       =last
  =imaginal>
)
|#

(p eval-opp-last-offer-2
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  =imaginal>
      isa               game
      opp-first-offer    =first
      opp-last-offer     =last
      - opp-last-offer     coop
      - opp-last-offer     aggr

  =retrieval>
      isa          last-offer
      strategy      =strat
==>
  =retrieval>
      player       =opponent
  -retrieval>
  =imaginal>
      opp-last-offer     =strat
      ;my-strategy        nil
      opp-strategy       nil
      my-first-offer     nil
      my-last-offer      nil
  -imaginal>
  +retrieval>
      isa               game
      opponent          =opponent
      ;my-strategy       coop
)

(p pick-coop-1
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                 game
       opp-first-offer      coop
       opp-last-offer       coop
==>
  !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  coop
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)

(p pick-coop-2
   =goal>
      isa          state
      status       active
      opponent     =opponent
```

```
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa         game
       opp-first-offer       aggr
       opp-last-offer        coop
==>
  !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  coop
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)

(p pick-coop-3
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                 game
       opp-first-offer       coop
       opp-last-offer        aggr
==>
 !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  coop
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)


(p pick-aggr
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                 game
       ;my-strategy          coop ;temp
       opp-first-offer       aggr
       opp-last-offer        aggr
==>
  !output! aggressive
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  aggr
  -goal>

)

#|
(p pick-aggr-2
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
```

```
  =retrieval>
       isa                game
       ;my-strategy          coop ;temp
       opp-first-offer       aggr
       opp-last-offer        coop
==>
  !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  aggr
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)

(p pick-aggr-3
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                game
       ;my-strategy          coop
       opp-first-offer       coop
       opp-last-offer        aggr
==>
  !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  aggr
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)
|#
#|
(p pick-neutral-1
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                game
       ;my-strategy          coop
       opp-first-offer       coop
       opp-last-offer        aggr
==>
  !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  neutral
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)

(p pick-neutral-2
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                game
       ;my-strategy          coop ;temp
```

```
       opp-first-offer       aggr
       opp-last-offer        coop
==>
  !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  neutral
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)
|#

;;Use the following productions if you want the model to randomly select its strategy
#|
(p pick-coop-4
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                 game
       opp-first-offer       aggr
       opp-last-offer        aggr
==>
  !output!  cooperative
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  coop
  -goal>
  ;!eval! (setf *coop* (+ 1 *coop*))
)

(p pick-aggr-5
   =goal>
      isa          state
      status       active
      opponent     =opponent
      outcome      =outcome
      points       =points
  ?imaginal>
      buffer       empty
  =retrieval>
       isa                 game
       ;my-strategy          coop ;temp
       opp-first-offer       coop
       opp-last-offer        coop
==>
  !output! aggressive
  +imaginal>
      isa          game
      opponent     =opponent
      my-strategy  aggr
  -goal>

)
|#
)
```